Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Teemu Vilkman

OpenShift DevOps-ohjelmistokehitysalustana

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

8.5.2019

Tekijä Otsikko Sivumäärä Aika	Teemu Vilkman OpenShift DevOps-ohjelmistokehitysalustana 29 sivua + 1 liitettä 8.5.2019
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tietotekniikan koulutusohjelma
Ammatillinen pääaine	Ohjelmistotekniikka
Ohjaajat	Lehtori Simo Silander

Työn tarkoituksena on testata ja arvioida Red Hatin OpenShift-ohjelmistokehitysalustaa ja sen ominaisuuksia DevOps-tyylisessä ohjelmistokehityksessä. Työssä käydään läpi DevOps-ideologian perusteet sekä teknologioita kuten Docker ja Kubernetes, joita OpenShift hyödyntää.

Työssä testataan yksinkertaisen valmiin PHP-sovelluksen ja sen SQL-tietokannan pystyttämistä OpenShift Online -alustalla käyttäen hyväksi OpenShift-katalogin tarjoamia PHP- ja MySQL Docker -imageja, jotka sisältävät valmiit ajoympäristöt.

Lisäksi työssä julkaistaan sovellus internetiin ja tutustutaan OpenShiftin automaattisiin ylläpito-ominaisuuksiin sekä A/B-testaukseen.

Työssä saatiin hyvä kokonaiskuva OpenShift-ohjelmistokehitysalustan ominaisuuksista ja ja soveltuvuudesta DevOps-ohjelmistokehitykseen.

Avainsanat	OpenShift, DevOps, Kubernetes, Docker

Author Title Number of Pages Date	Teemu Vilkman OpenShift as a DevOps Software Platform 29 pages + 1 appendices 8.5.2019
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Software Engineering
Instructors	Senior Lecturer Simo Silander

The aim of this thesis was to evaluate Red Hat's OpenShift platform for DevOps style software development. The thesis covers DevOps fundamentals and the technology stack that OpenShift utilizes such as Docker and Kubernetes.

The work demonstrates how to setup a PHP project with an SQL-database using Docker images from the OpenShift catalogue, which contain environments for running a MySQL database and PHP application.

The software was also published to the Web and health checks were setup for automatic maintenance. Finally, an A/B test is setup.

The project was a good overview of OpenShift for DevOps style software development.

Keywords	OpenShift, DevOps, Kubernetes, Docker

Sisällys

Lyhenteet

1 Johdanto 1

2 DevOps 1

2.1 DevOpsin kolmijakoinen ajatusmalli 3

2.1.1 Systeemiajattelu 3

2.1.2 Palautesilmukoiden vahvistus 3

2.1.3 Jatkuva oppiminen ja toiminnan kehittäminen 3

2.2 Jatkuva integraatio ja toimitus 4

3 Käytetyt teknologiat 4

3.1 Docker ja kontitus 4

3.1.1 Kubernetes ja orkestrointi 5

3.1.2 OpenShift 6

4 Sovelluksen rakentaminen, ajaminen ja julkaisu 7

4.1 Projektin pystytys ja sovelluksen koonti 8

4.1.1 OpenShift Online -käyttöliittymän päänäkymät 8

4.1.2 Sovelluksen pystytys PHP-imagesta 10

4.1.3 Tietokannan lisääminen projektiin 15

4.2 Sovelluksen ajaminen 16

4.2.1 Ajokonfiguraation tekeminen 16

4.2.2 Konfiguraation injektointi konttiin 17

4.3 Sovelluksen julkaisu 21

4.3.1 Servicet 21

4.3.2 Reititys 23

5 Sovelluksen ylläpito ja A/B-testaus 24

5.1 Health checkit 24

5.2 A/B-testaus 26

6 Yhteenveto 28

Lähteet 29

Lyhenteet ja käsitteet

A/B-testaus	A/B-testauksella tarkoitetaan kahden eri version samasta tuotteesta (versio A ja versio B) satunnaistettua testaamista, jossa testataan, kumpi versio on parempi.
CI	Continuous Integration eli uuden lähdekoodin jatkuva integroiminen sovelluksen kehityksessä.
CD	Continuous Delivery eli sovelluksen jatkuva toimitus tuotantoympäristöön.
DevOps	Development and Operations, kehitys ja ylläpito. Ohjelmistokehityksen ideologia.
Docker	Ohjelmien kontituksen ja konttien ajamisen työkalu.
Git	Versionhallintatyökalu.
GitLab	Versionhallinta-alusta.
Jenkins	CI/CD-alusta.
Kubernetes	Googlen kehittämä orkestrointialusta.
MVC	Model-view-controller. Usein käytetty sovellusarkkitehtuuri/suunnittelumalli.
MVP	Minimum Viable Product eli pienin mahdollinen toimiva tuote.
OpenShift	Red Hatin kehittämä ohjelmistoalusta.
PaaS	Platform-as-a-Service, palvelualustan ulkoistaminen pilvipalveluun.
Pod	Kubernetesin ajettava kontti tai konttikokoelma.
YAML	YAML Ain’t Markup Language. Merkistökieli.

1 Johdanto

Tässä opinnäytetyössä pyrittiin testaamaan Red Hatin OpenShift-ohjelmistosalustaa DevOps-tyylisessä kehityksessä. Työssä käydään läpi lyhyesti, mitä DevOps on, ja pohjustetaan OpenShiftin hyödyntämiä teknologioita.

Aluksi valittiin Red Hatin tarjoamista OpenShift PaaS (Platform-as-a-service) -vaihtoehdoista alustaksi OpenShift Online. Sitten valittiin tarkoitukseen sopiva, tarpeeksi yksinkertainen PHP-sovellus pystytettäväksi OpenShift-alustalle.

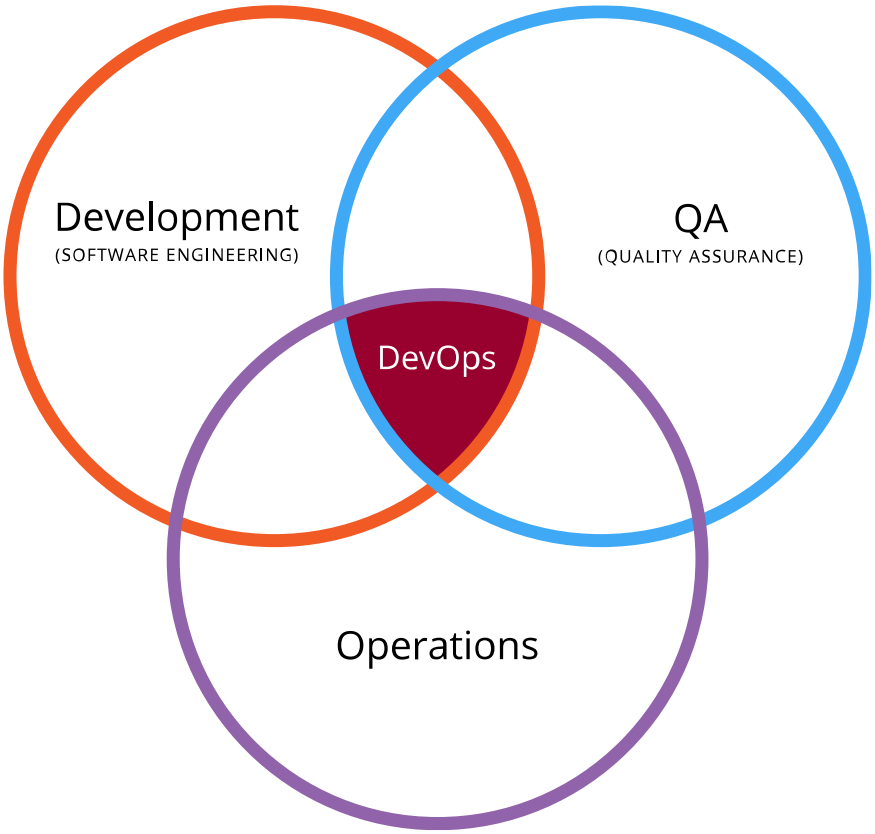
Seuraavaksi OpenShift Onlineen pystytettiin sovellukselle ajoympäristö hyödyntäen PHP-imagea OpenShiftin valmiiden sovellusympäristöjen katalogista. Sovelluksen lähdekoodi kytkettiin OpenShiftiin GitLab-versionhallinta-alustan kautta sen Webhook-toiminnallisuutta käyttäen. Sovellus julkaistiin myös internetiin käyttäen OpenShiftin liikenteen reititykseen tarjoamia työkaluja.

Lopuksi testattiin sovelluksen automaattiseen ylläpitoon tarkoitettuja, OpenShiftin hyödyntämän, Kuberneteksen tarjoamia health checkejä, sekä testattiin vielä, kuinka tehdä nk. A/B-testausta OpenShiftiä hyväksikäyttäen.

2 DevOps

DevOps käsitteenä on noin 10 vuotta vanha ja sai alkunsa Gentissä Belgiassa Patrick Deboisin järjestämässä tapahtumassa DevOpsDays. Tapahtuman innoitti aiemmin samana vuonna 2009 Flickr:in työntekijöiden John Allspawn ja Paul Hammondin pitämä esitys ”10 Deploys per Day: Dev and Ops Cooperation at Flickr” eli löyhästi suomennettuna ”10 tuotantoon ajoa päivässä: Kehityksen ja tuotannon kohtaaminen Flickr:ssä”. Kyseisen esityksen paikan päällä nähneet ovat kuuleman mukaan tienneet heti, että kyse oli mullistavasta ajatuksesta. [1, s. 5.]

DevOps on ohjelmistotekniikan ideologia ja toimintamalli, jossa ketteryys on avainasemassa ja perustuukin ketterän kehityksen sekä Toyotan aikoinaan kehittämän Lean-filosofian pohjalle [1, s. 4.] DevOps termi tulee sanoista *development* ja *operations*. Eli puhutaan kehityksen ja tuotannon (ylläpidon) kohtaamisesta. Pyrkimyksenä on luoda mahdollisimman nopeasti tomiva versio ohjelmistosta (voidaan kutsua pienin toimiva tuote tai engl. MVP = Minimum Viable Product). Ohjelmiston kehittyessä pyritään integroimaan pienetkin uudet ominaisuudet siihen ja keräämään palaute nopeasti sen sijaan, että uudet ominaisuudet toteutettaisiin isoina yksittäispäivityksinä. Kuvassa 1 on kuvattu, kuinka DevOpsissa kohtaavat ohjelmistokehitys (engl. development), laadunvalvonta (engl. quality assurance) ja ylläpito (engl. operations). Ohjelmistokehitys ja jatkuva integrointi liittyvät termin Dev-puoleen, kun taas Ops-puolen osuus on jatkuvassa toimituksessa, sekä konttiteknologiassa ja orkestroinnissa, joista lisää luvuissa 3 ja 4.



Kuva 1. Venn-diagrammi kuvastaa ohjelmistotuotannon osa-alueiden limittymistä DevOpsissa. (https://en.wikipedia.org/wiki/DevOps#/media/File:Devops.svg CC-BY 3.0)

2.1 DevOpsin kolmijakoinen ajatusmalli

2.1.1 Systeemiajattelu

DevOpsissa pyritään ajattelemaan kehitysprosessia eräänlaisena systeeminä, joka ei voi toimia tehokkaasti kokonaisuutena, jos jokin yksittäinen osa hidastelee [1].

Käytännössä tällä voidaan tarkoittaa esimerkiksi mahdollisimman monen ohjelmistotuotannon prosessin automatisointia. Tarkoitus on maksimoida nopeus, jolla ohjelmisto saadaan kehityksestä tuotantoon, kuitenkaan laadusta tinkimättä.

2.1.2 Palautesilmukoiden vahvistus

DevOpsissa palautesilmukalla (engl. feedback loop) voidaan tarkoittaa esimerkiksi loppukäyttäjän antamaa palautetta ohjelmistokehittäjälle, mutta myös automaatiotestauksen antama palaute kehittäjälle lasketaan palautesilmukaksi ja on hyvä esimerkki lokalisoidummasta palautesilmukasta, joita DevOpsissa pyritään rakentamaan. [1, s. 27.]

Automatisaatio auttaa myös ylläpitämään ohjelmiston laatua esim. automaatiotestauksen keinoin, jolla ohjelman toiminnot voidaan testata. Yleisiä testitapoja on yksikkötestaus, jossa ohjelman eri osia testataan, sekä staattinen analyysi, jossa tutkitaan itse koodin laatua ajamatta sitä. Näitä erilaisia testejä voidaan toteuttaa niin, että kun versionhallintaan tulee uusi versio ohjelmasta, niin tapahtuma aloittaa ketjun, jossa ohjelmalle ajetaan ensin testit. Sitten se paketoidaan ja lopuksi asennetaan testipalvelimelle. Näin ei hukata aikaa ohjelmapaketin rakentamiseen ja manuaaliseen asennukseen. Tällaista kokonaisuutta kutsutaan CI/CD-putkeksi tai liukuhihnaksi (engl. pipeline).

2.1.3 Jatkuva oppiminen ja toiminnan kehittäminen

Kolmas DevOpsin perusajatus on täysin teknologiasta riippumaton, jatkuvan kehittymisen ja oppimisen kulttuurin vaaliminen. Tämä tarkoittaa sitä, että harkittuja riskejä tulee ottaa, ja virheitä saa tehdä. Virheistä tulee oppia sekä henkilökohtaisella että organisaatiotasolla. [1, s.13.]

2.2 Jatkuva integraatio ja toimitus

Ohjelmistojen kehityksessä on perinteisesti julkaistu vain harvoin uusia isoja versioita, joiden kehittäminen kestää ja joiden testaaminen tapahtuu vasta pitkän kehitysajan jälkeen. Ketterässä ohjelmistokehityksessä tällaisesta pyritään eroon mm. jatkuvaa integraatiota käyttäen.

Jatkuvalla integraatiolla (engl. Continuous Integration) tarkoitetaan ohjelmiston kehityksen aikana tehtävää jatkuvaa koostamista aina, kun uutta koodia valmistuu. Käytännössä tämä voidaan toteuttaa Git-versionhallinnan avulla siten, että kun versionhallintaan tulee uutta koodia, niin laukaistaan automaattisesti ohjelman koonti käyttäen webhookia.

Jatkuvalla toimituksella puolestaan tarkoitetaan uusien sovellusversioiden jatkuvaa toimitusta ympäristöstä toiseen käyttäen hyödyksi esimerkiksi automaatiota ja konttiteknologiaa, josta kerrotaan lisää seuraavassa luvussa. [3. s. 2.]

3 Käytetyt teknologiat

3.1 Docker ja kontitus

Perinteisesti ohjelmien jakaminen on tapahtunut ajettavien tiedostojen kautta, ja uuteen kohdeajoympäristöön on pitänyt asentaa myös ohjelmiston vaatima käyttöjärjestelmä ja ohjelmakirjastot. Tämän kaltainen menettely tuottaa paljon työtä, etenkin jos sama ohjelmisto on asennettava useampaan ympäristöön. Docker on avoimen lähdekoodin projekti, jonka pyrkimyksenä on helpottaa ohjelmien jakamista ja ajamista missä tahansa ympäristössä. Kontitus on osa DevOpsin ”operations” -puolta.

Dockerin ratkaisu ongelmaan on ohjelmistojen ”kontittaminen”, eli ohjelmistojen pakkaaminen käyttöjärjestelmää myöten valmiiksi paketiksi, jolloin niiden ajaminen missä tahansa ympäristössä helpottuu ja säästytään palvelimen, kirjastojen ja ympäristöjen asennuksilta. Pakattua käyttöjärjestelmää ja sovellusta ajetaan omassa hiekkalaatikossaan tai nimiavaruudessaan, jossa ajoympäristö on valmiiksi konfiguroitu. Tämä on mahdollista käyttämällä niin sanottua virtualisointia, joka hyödyntää Linux-kernelin (ytimen)

resurssieneristystä. Näin palvelimelle ei tarvitse asentaa muuta kuin jokin Dockeria tukeva käyttöjärjestelmä ja itse Docker, joka pyörittää virtualisoitua ympäristöä kontista.

Dockerilla paketoitua sovellusta kutsutaan Docker-imageksi. Docker imaget ovat nykyään erittäin suosittu tapa jakaa ohjelmistoja, koska ne säästävät palvelimen ajoympäristön pystyttämisen vaivalta. Docker Hub -palvelussa (https://hub.docker.com) on kirjoitushetkellä hyödynnettävissä yli satatuhatta imagea. Todella suosittuja, yli kymmenen miljoonaa kertaa ladattuja imageja ovat esimerkiksi Ubuntu-käyttöjärjestelmän ja PostreSQL-relaatiotietokannan sisältävät imaget. Dockerin virallisia imageja ylläpidetään erittäin aktiivisesti, ja ne takaavat niiden päälle pystytettäville ohjelmistoille ajankohtaiset versiot ohjelmistoista ja käyttöjärjestelmistä. Käytännössä Docker-paketoinnin yhteydessä voidaan konfiguraatioon määritellä käytettävän aina uusinta tarjolla olevaa imagea.

Projektissa hyödynnettiin OpenShiftin tarjoamaa Docker-imagekatalogia, joka sisältää valmiita pohjia monenlaisten sovellusten ajamiseen. Täysi katalogi löytyy osoitteesta https://access.redhat.com/containers/.

3.2 Kubernetes ja orkestrointi

Orkestroinnilla tarkoitetaan ohjelmistomaailmassa kokonaisten työnkulkujen tai prosessien automatisointia ja sen pyrkimyksenä on virtaviivaistaa ohjelmistojen jakelua. Se on ikään kuin äärimmilleen vietyä automatisaatiota, jossa käytännössä kaikki toistettavat prosessit pyritään automatisoimaan mahdollisimman laajasti [2]. Konttien tulo ohjelmistomaailmaan on tuonut tarpeen myös niiden orkestroinnille eli konttien automaattiseen hallintaan ja palveluiden skaalaamiseen.

Konttien orkestroinnilla tarkoitetaan pohjimmiltaan resurssien automaattista allokointia. Konkreettisena esimerkkinä voisi olla tavallinen palvelinklusteri, johon on asennettu kontista pyörivä sovellus. Sovelluksen ajaminen vaatii prosessoriaikaa ja RAM-muistia. Nämä ovatkin hyvä ja tyypillinen esimerkki automaattisesti allokoituvasta resurssista. Orkestrointi abstrahoi käyttäjältä virtuaalikoneiden resurssien automaattisen tasapainon (eng. Load balancing).

Kubernetes on Googlen avoimen lähdekoodin konttien orkestrointialusta, jota käytetään myös OpenShiftissä orkestrointitasona, jonka päälle OpenShift tuo lisää ominaisuuksia kuten web-liikenteen reitityksen alustalla pyöriviin palveluihin ja valmiin kokoelman Docker-kontteja eri tarkoituksiin (OpenShift Container Registry).

Kubernetesin pienin ajettava yksikkö on niin kutsuttu *Pod.* Yksi Pod voi sisältää yhden tai useampia ajettavia kontteja, joilla on jaettu verkko ja levykapasiteetti sekä yhteinen ajokonfiguraatio. Podin sisältö ajetaan aina yhtenä kokonaisuutena - sisälsi se kuinka monta konttia tahansa. Yleisimmässä käyttötapauksessa Pod kapseloi kuitenkin vain yhden kontin.

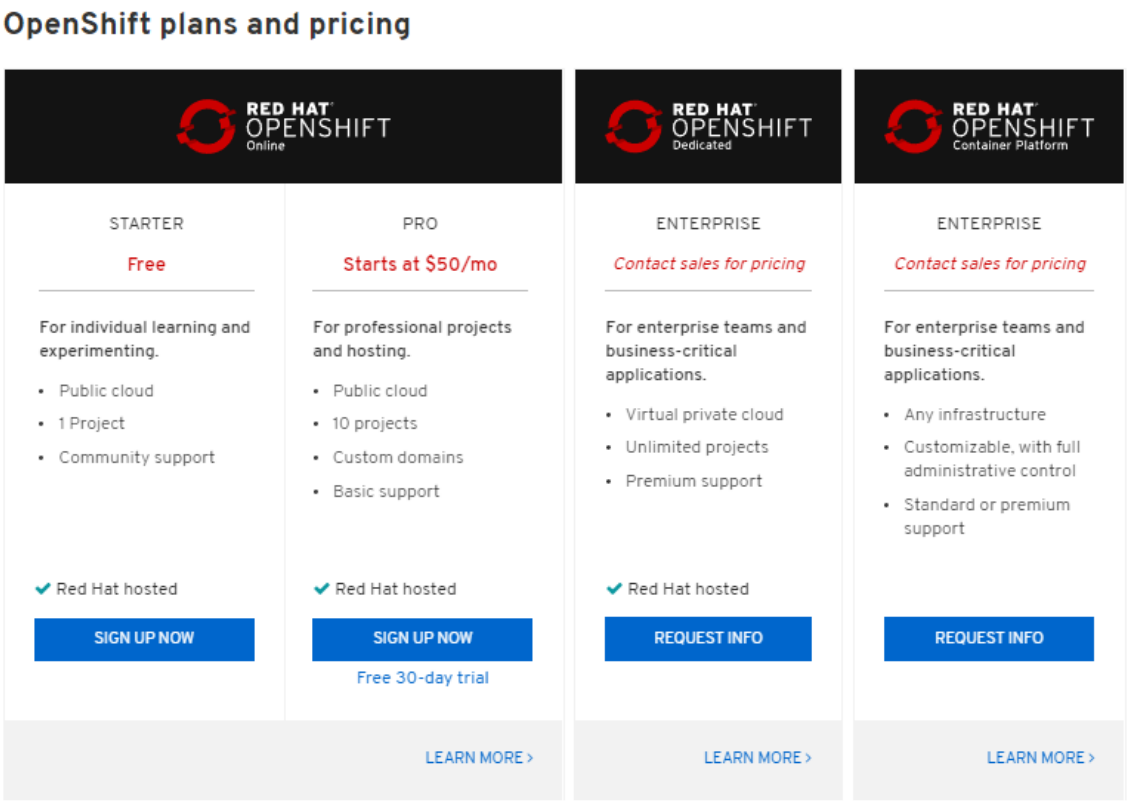
3.3 OpenShift

OpenShift on Red Hatin tarjoama alusta, joka hyödyntää edellisissä luvuissa esiteltyjä teknologioita tarjoten ne yhtenä ohjelmistoalustana, joko pilvipalveluna tai avoimen lähdekoodin version omalle palvelinklusterille. OpenShiftin testaaminen on tämän työn päätavoite.

OpenShiftin avoimen lähdekoodin versiota kutsutaan OKD:ksi (Nimi vaihtui kesken tämän työn tekemisen ja oli ennen syksyä 2018 OpenShift Origin). OpenShift-alustan pystyttäminen alusta loppuun on melko vaativa operaatio, jota varten on pystytettävä minimivaatimukset täyttävä palvelinklusteri.

Red Hat tarjoaa OpenShiftistä useita erilaisia palveluna tarjottavia vaihtoehtoja (PaaS, Platform-as-a-Service), joista OpenShift Online on kevyin, ja ainut, jota voi käyttää ilmaiseksi, joskin melko rajatuilla resursseilla. Muita vaihtoehtoja kirjoitushetkellä ovat OpenShift Dedicated, ja OpenShift Container Platform, jotka on tarkoitettu järeämpään yrityskäyttöön (ks. Kuva 2).

Tätä työtä aloitettaessa kokeiltiin tehdä OKD:n asennusta tyhjälle virtuaalipalvelimelle, mutta se osoittautui melko haastavaksi ja aikaa vieväksi, ja työssä haluttiin keskittyä OpenShiftin käyttämiseen, eikä sille sopivan palvelimen konfiguroimiseen. Tästä syystä tämän työn tarpeisiin valittiin lopulta OpenShift Online.



Kuva 2. OpenShift PaaS-vaihtoehdot. (Kuvankaappaus osoitteesta https://www.openshift.com/products/pricing/ 20.2.2019)

4 Sovelluksen rakentaminen, ajaminen ja julkaisu

Tässä luvussa käydään läpi OpenShift projektin teko, sovelluksen lähdekoodin tuonti ja koonti, sekä sovelluksen ajaminen. Lisäksi vielä tehdään sovelluksen julkaisu internettiin.

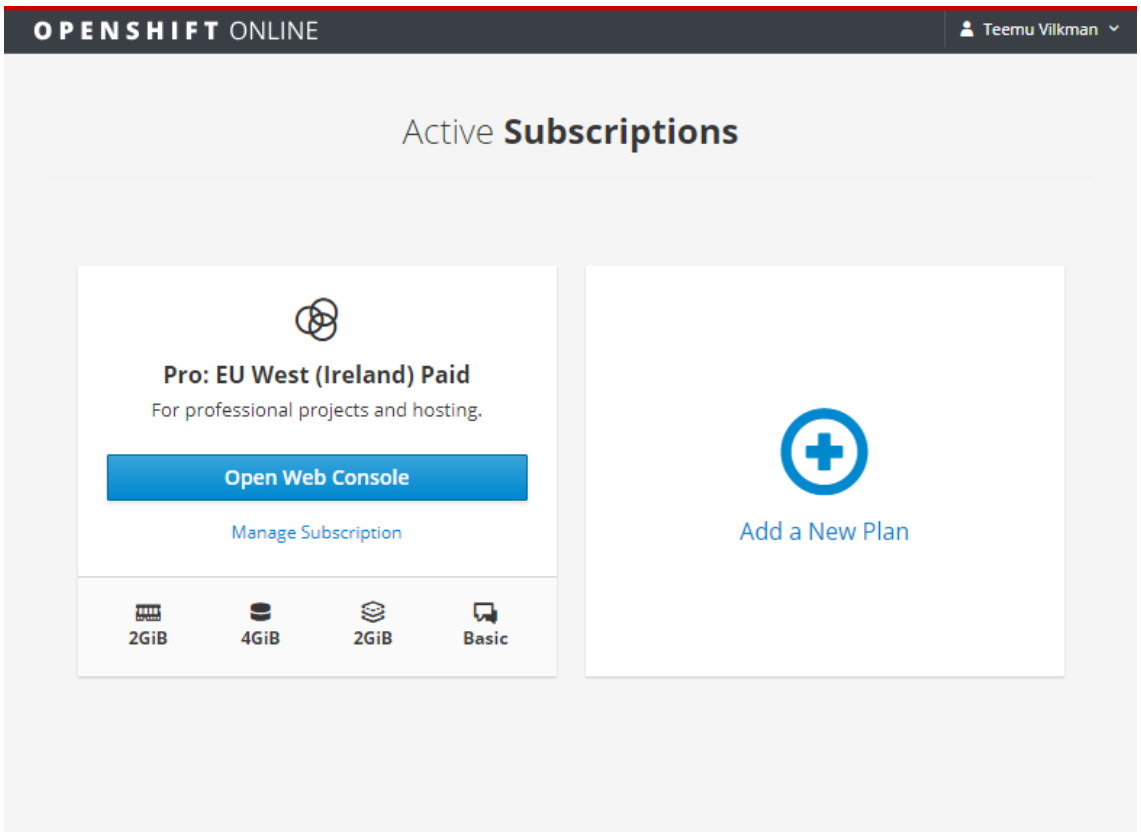
Kuva 3. Projektin työnkulku OpenShiftissä. Korostettuna meneillään oleva vaihe.

4.1 Projektin pystytys ja sovelluksen koonti

Tässä luvussa tarkastellaan projektin pystyttämistä OpenShiftiin ja käydään läpi sen tärkeimmät käyttöliittymät. Myös sovelluksen koonti käsitellään. OpenShiftin käyttäminen onnistuu myös omalta koneelta komentorivityökalua nimeltä OC tools käyttäen edellyttäen että se on asennettu.

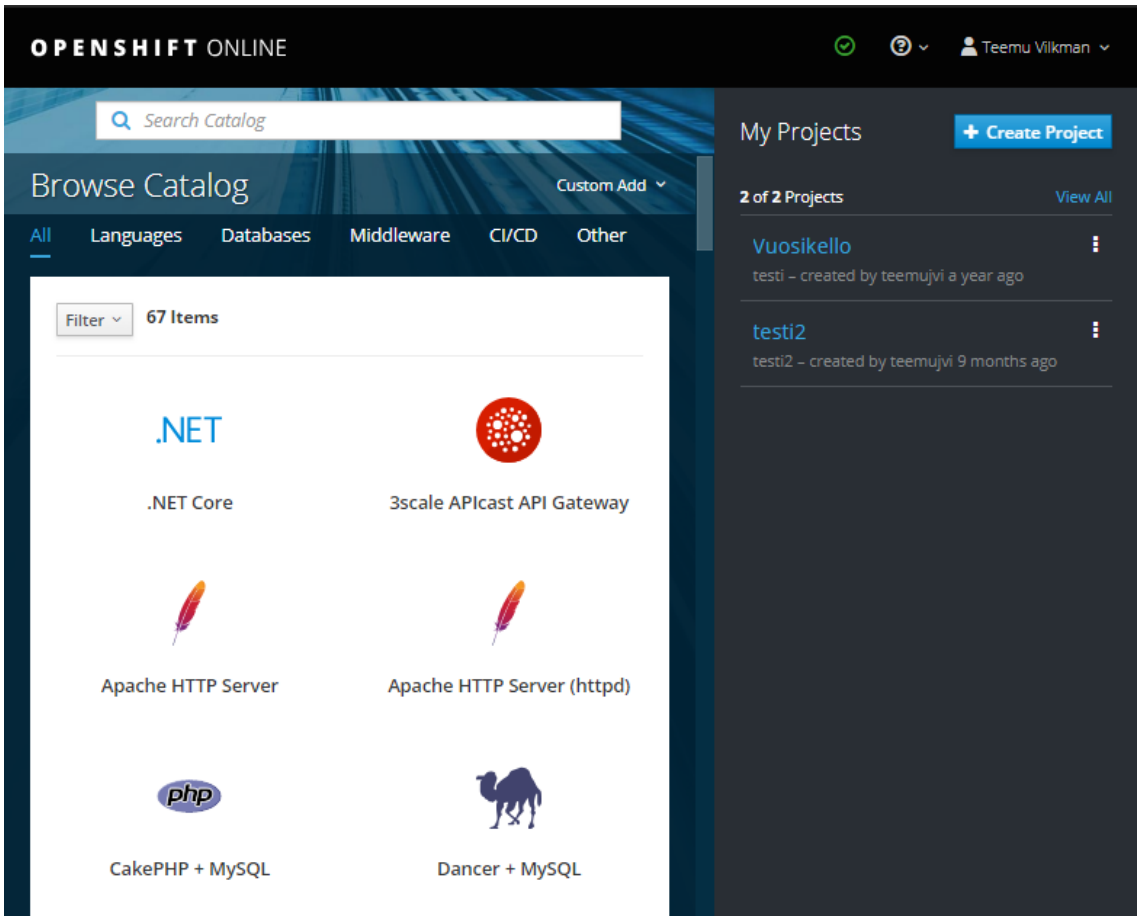
4.1.1 OpenShift Online -käyttöliittymän päänäkymät

OpenShift Onlinessa on muutama pääkäyttöliittymä, jotka esitellään tässä luvussa. Kirjautuessa Red Hat -tunnuksella palveluun, avautuu näkymä, josta valitaan voimassa oleva OpenShift-tilaus (kuva 4):



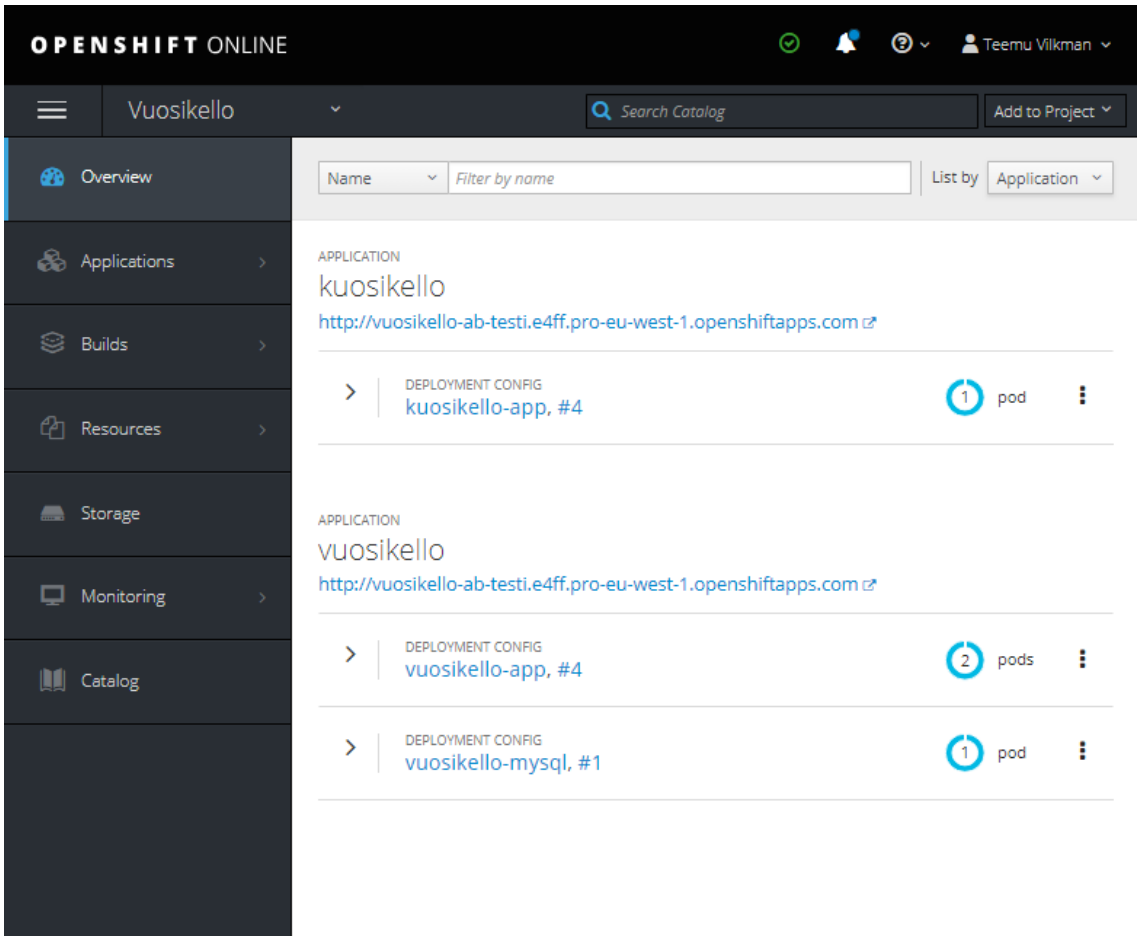
Kuva 4. ”Active Subscriptions” -näkymä OpenShift Online -käyttöliittymässä (Kuvankaappaus OpenShift Online -käyttöliittymästä 20.3.2019)

Tilauksen valittua tullaan konsolinäkymään, jossa näytetään olemassa olevat projektit tai luodaan uusi käyttäen jotain OpenShiftin tarjoamista imageista (kuva 5):



Kuva 5. OpenShift Online -näkymä tilauksen valitsemisen jälkeen (Kuvankaappaus OpenShift Online -käyttöliittymästä 20.3.2019)

Kun projekti on valittu tai tehty uusi, päästään tärkeimpään näkymään, josta säädetään projektin ja sovelluksen asetuksia (kuva 6):



Kuva 6. Projektinäkymä OpenShift Onlinessa (Kuvankaappaus OpenShift Online -käyttöliittymästä 20.3.2019)

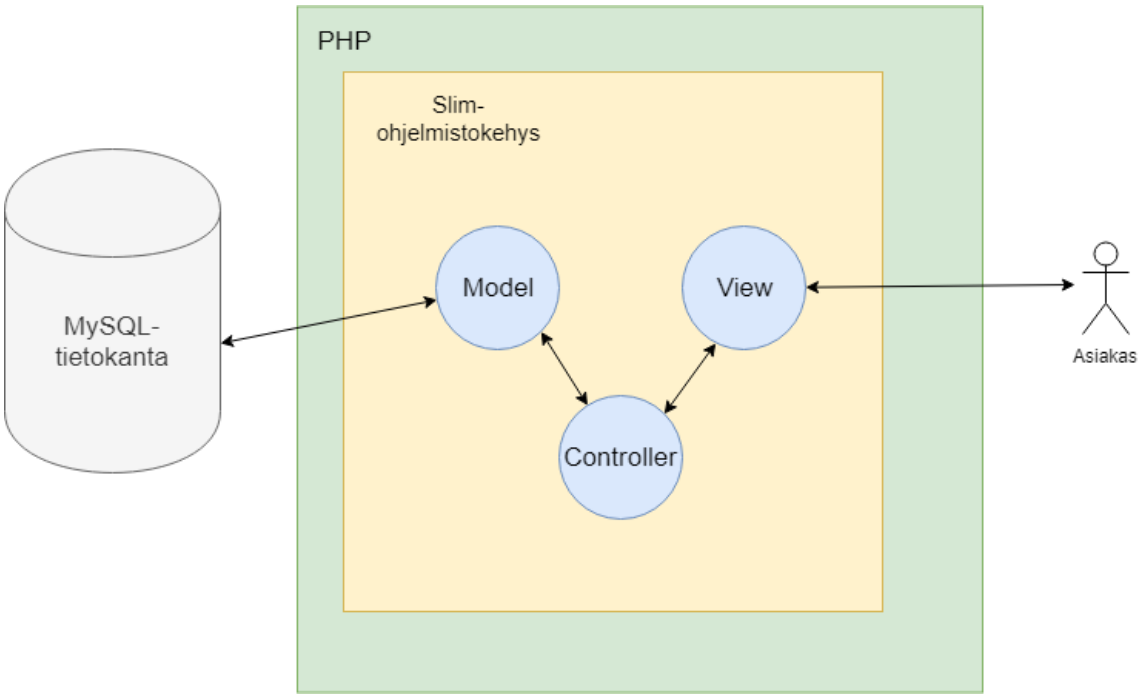
Projektinäkymästä käsin tehdään suurin osa työn aikana tehtävistä projektiin liittyvistä toimista.

4.1.2 Sovelluksen pystytys PHP-imagesta

Työtä varten haluttiin valita melko yksinkertainen Web-sovellus, jolla kuitenkin on tietokanta, ja jossa ei käytetä sensitiivistä tietoa. Tarkoitukseen valittiin Metropolian Vuosikello PHP-sovellus, joka on eräänlainen vuosikalenteri. Sovellus käyttää minimalistista Slim-sovelluskehystä, joka on tarkoitettu kevyiden PHP-sovellusten luontiin.

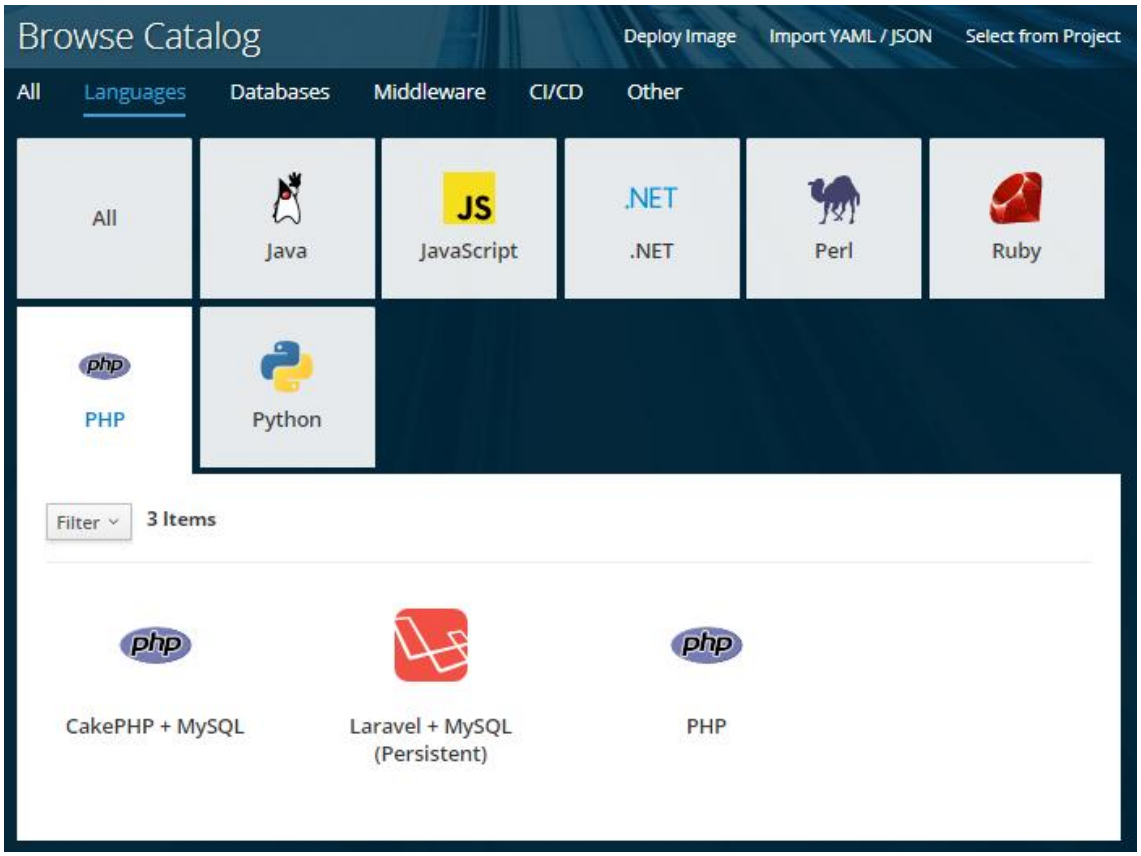
Kuva 7. Kuvankaappaus Vuosikello sovelluksen -käyttöliittymästä (20.2.2019)

Slim-ohjelmistokehys noudattaa MVC-sovellusarkkitehtuurimallia, eli se koostuu osista Model (malli), joka edustaa tietokantakerrosta, View (näkymä) eli loppukäyttäjän käyttöliittyymän renderöivä osa, ja Controllerista (ohjain) joka ohjaa tapahtumia näiden välillä (kuva 8.)



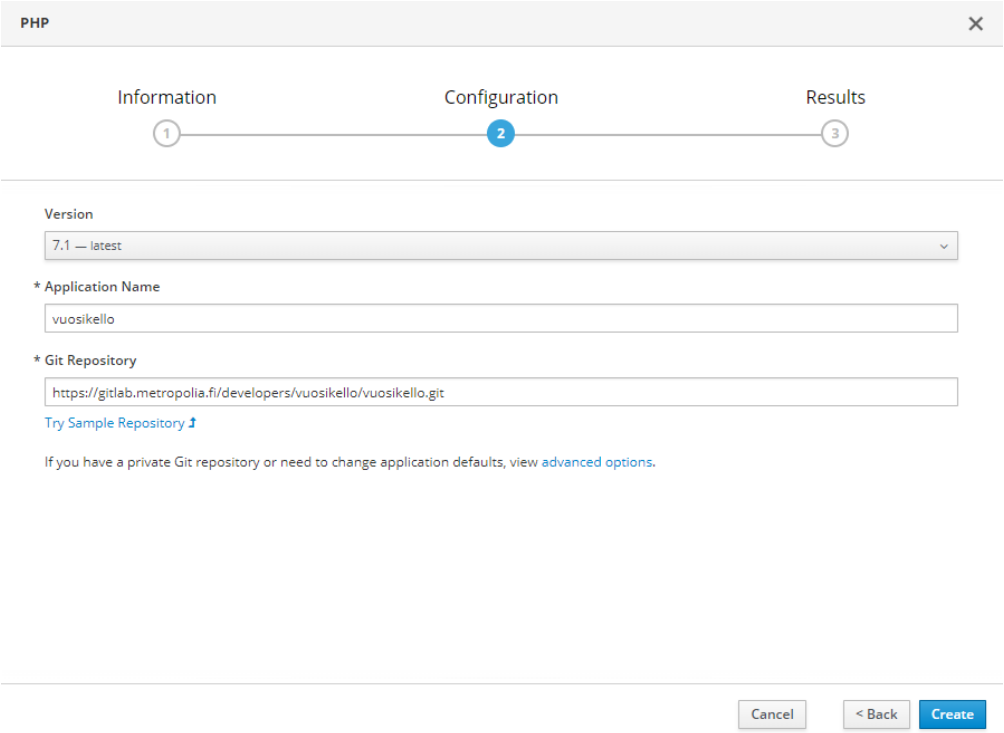
Kuva 8. Vuosikello-sovelluksen arkkitehtuuri.

OpenShiftin käyttöliittymästä luotiin uusi projekti, jonka jälkeen käytettiin hyväksi OpenShiftin Docker image -katalogia, josta löytyy vaihtoehto ”PHP”, joka sisältää valmiin alustan PHP-sovellukselle:



Kuva 9. Kuvassa OpenShiftin imagekatalogin PHP-vaihtoehdot. (Kuvankaappaus OpenShift Online käyttöliittymästä 20.2.2019)

Kun sopiva pohjaimage on valittu, syötetään projektin perustiedot seuraavassa valinta-dialogissa (kuva 10):



Kuva 10. Imagen valinnan jälkeisessä projektinluontinäkymässä annetaan Git-repositorion osoite ja perustiedot projektille. (Kuvankaappaus OpenShift Online -käyttöliitymästä 8.8.2018)

OpenShift käyttää sovelluksen koontiin ns. ”Source-to-Image” -koontistrategiaa, jossa valmiiseen imageen injektoidaan sovelluksen lähdekoodi ja koostetaan sitten uusi image.

OpenShiftissä sovelluksen (app) luomiseen tarvitaan vain Gitlab-palvelussa olevan koodirepositorion osoite, mutta jotta sovelluksen lähdekoodin voi ladata, on OpenShiftille annettava pääsy repositorioon, sillä se ei ole julkinen, vaan kyse on suljetusta lähdekoodista. Pääsy toteutettiin käyttämällä nk. Access Tokenia, joka luotiin Gitlabin tilin asetuksista. OpenShiftin puolella valittiin tämän jälkeen ”Resources” -osion alta ”Secrets” eli salaisuudet, jonne lisätään Gitlabin generoima token (kuva 11). Tämän jälkeen token lisättiin vielä vuosikellon koontiasetukset sisältävään YAML-tiedostoon, jonka OpenShift loi automaattisesti. OpenShift käyttää YAML-merkistökieltä lähes kaikkien asetusten tallentamiseen. YAML-tiedostojen sisällöt saa yleisesti auki ”Actions”-painikkeen, joka on käyttöliittymän oikeassa reunassa, kontekstivalikosta ”Edit YAML”. Koontiasetukset sisältävä YAML-tiedosto on esimerkkinä kokonaisuudessaan liitteessä 1.

Edit Build Config vuosikello

```
19          name: 'vuosikello:latest'
20      postCommit: {}
21      resources: {}
22      runPolicy: Serial
23      source:
24        git:
25          ref: openshift
26          uri: 'https://gitlab.metropolia.fi/developers/vuosikello/vuosikello.git'
27        sourceSecret:
28          name: gitlab-token
29        type: Git
30      strategy:
31        sourceStrategy:
32          from:
33            kind: ImageStreamTag
34            name: 'php:7.1'
35            namespace: openshift
36        type: Source
37      triggers:
38        - imageChange:
39            lastTriggeredImageID: >-
40              docker-registry.default.svc:5000/openshift/php@sha256:62c63fb7aeffd624f49c8ebf6c4a33addc824e44fd315ac2917753f783c633a3
41          type: ImageChange
42        - type: ConfigChange
43        - generic:
44            secret: a33545b3e13b8630
45          type: Generic
46        - github:
47            secret: 8d86e812f776d3ca
48          type: GitHub
49    status:
50      lastVersion: 5
51
```

Save Cancel

Kuva 11. Vuosikellon koontiasetukset sisältävä YAML-tiedosto, ympyröitynä lisätty Gitlab-token. Kuvankaappaus OpenShift Online -käyttöliittymästä 14.8.2018

Sovellus vaatii toimiakseen myös, että Web-palvelimen ympäristömuuttuja ”DocumentRoot” eli HTTP-juurihakemisto on asetettu oikein osoittamaan repositoriossa olevaan ”public”-kansioon. OpenShift etsii ympäristömuuttujia ”environment”-tiedostosta, polusta /<repositorio>/.s2i/environment. Sovellukseen tehtiin uusi koodihaara, jonne luotiin kyseinen tiedosto ja tiedostoon rivi: ”DOCUMENTROOT=/public” [4].

4.1.3 Tietokannan lisääminen projektiin

Sovelluksen MySQL-tietokanta lisättiin samaan tapaan valmiista MySQL Docker-imagesta ja varsinaisen kantarakenteen ja sovelluksen ajamiseen tarvittavan minimidatan lisäämiseen käytettiin SQL-skriptiä. Skripti lisättiin ”Resources” -osion alla olevaan ”Config Maps” -osioon. ConfigMapit mahdollistavat konfiguraatiotietojen injektoinnin ajetta-

vaan konttiin. ConfigMapit muistuttavat siis toiminnaltaan Secretejä, mutta ne on suunniteltu tukemaan vähemmän sensitiivisten tietojen tietojen hyödyntämistä kontissa pyörivässä ohjelmassa. ConfigMappeja voi luoda myös komentoriviltä komennolla *oc create configmap <configmapin nimi> [optiot].* Hyödyllinen optio komennolle on *--from-file=<tiedoston nimi>,* jonka avulla voi luoda myös kokonaisen kansion sisällöstä ConfigMapin [5].

4.2 Sovelluksen ajaminen

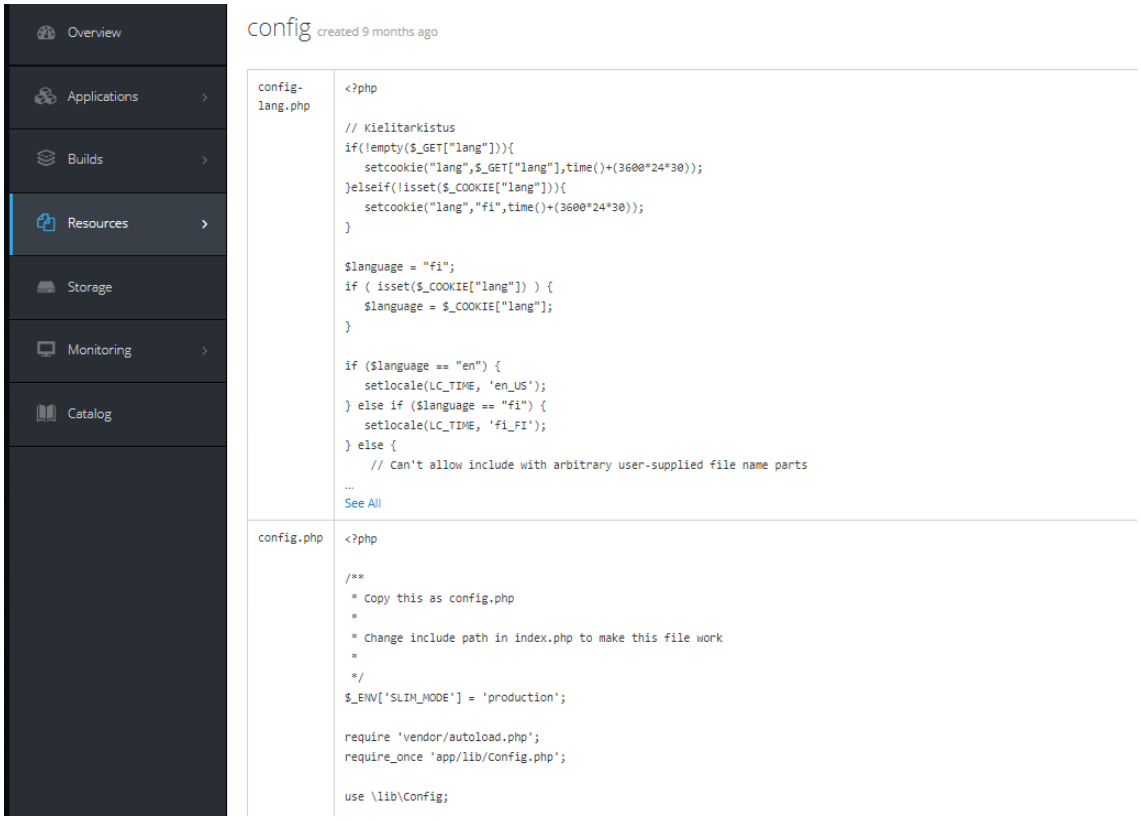
Sovellus on tässä vaiheessa saatu tuotua OpenShiftiin ja koottua onnistuneesti ja tässä luvussa sovellus laitetaan ajoon (kuva 12).

Kuva 12. Projektin työnkulku OpenShiftissä. Korostettuna meneillään oleva vaihe.

4.2.1 Ajokonfiguraation tekeminen

Kun sovellus on valmis OpenShiftin ajettavaksi, on aika tehdä ajokonfiguraatio (engl. deployment configuration). Sovelluksen ajaminen edellyttää myös sovelluskohtaista konfiguraatiota, joka laitetaan niin ikään ConfigMapiksi, kuten tietokannan luomiseen käytetty SQL-skripti edellisessä luvussa. Konfiguraatio koostuu kahdesta PHP-tiedostosta, jotka sisältävät kieleistykseen ja sovelluksen tietokantaan ja polkuihin liittyviä asetuksia. Vaikka konfiguraatiotiedostoja on useampi, niistä tehtiin yhteinen ConfigMap nimeltä ”config”. Konfiguraatiot löytyvät ”Resources”-osion alta, josta löytyy näkymä, joka näyttää myös tiedostojen sisällöt (kuva 13).

5



Kuva 13. Sovelluksen konfiguraatiot ”config” -nimisessä ConfigMapissa. (Kuvakaappaus OpenShift Online käyttöliittymästä 7.2.2019)

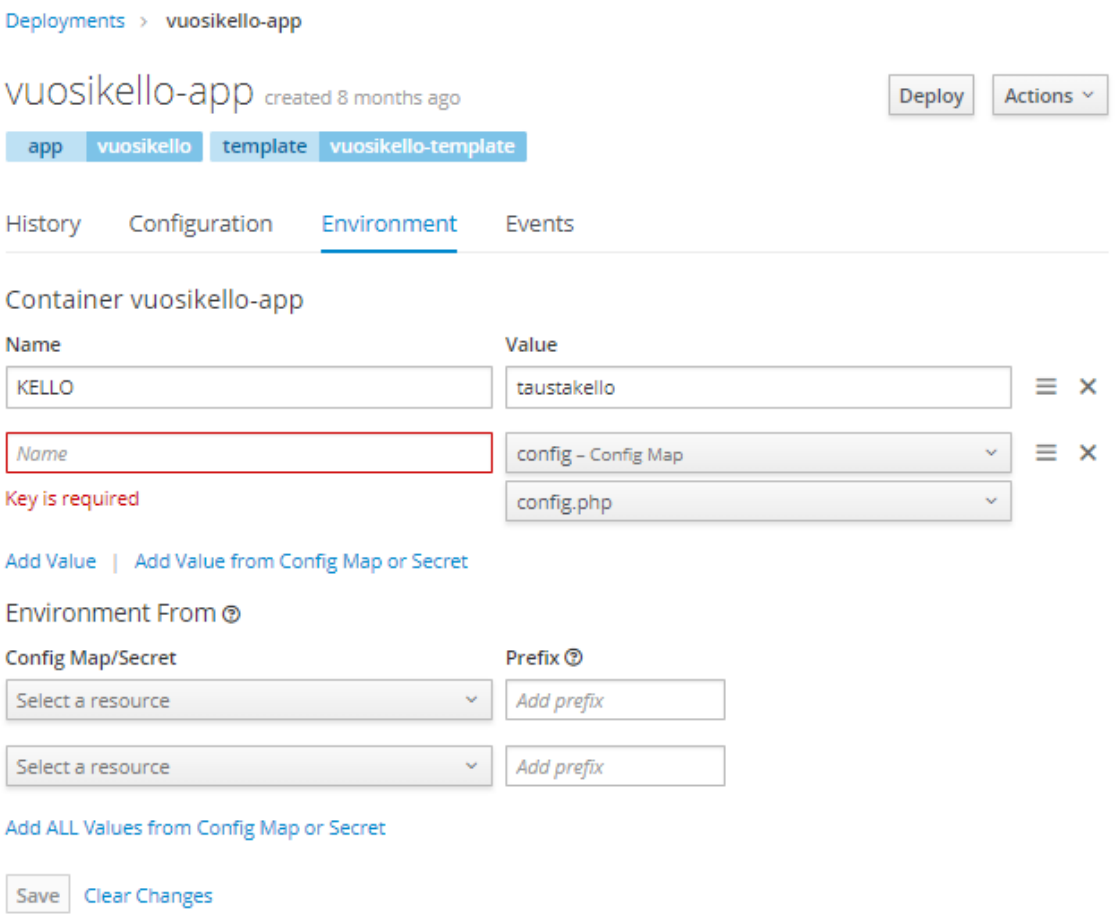
4.2.2 Konfiguraation injektointi konttiin

Kontti voi hyödyntää ConfigMappeja kahdella tavalla:

1) ympäristömuuttujaksi injektoituna

2) tiedostona kontin hakemistorakenteessa.

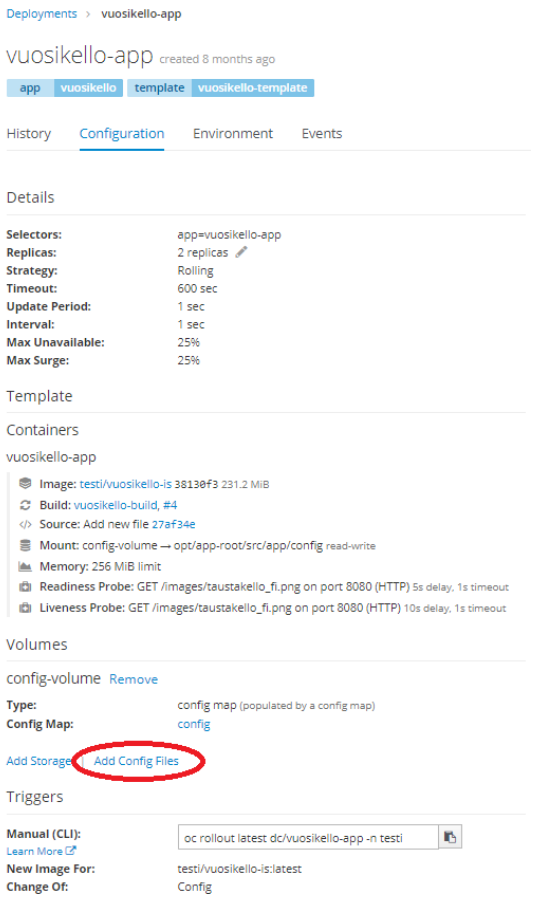
Ympäristömuuttujaksi injektoiminen tapahtuu ”Applications”-valikon kohdan Deployments alta, välilehdeltä Environment. Sitten klikataan ”Add Value from ConfigMap” ja valitaan pudotusvalikosta ”ConfigMap” ja sieltä injektoitava arvo (kuva 14).



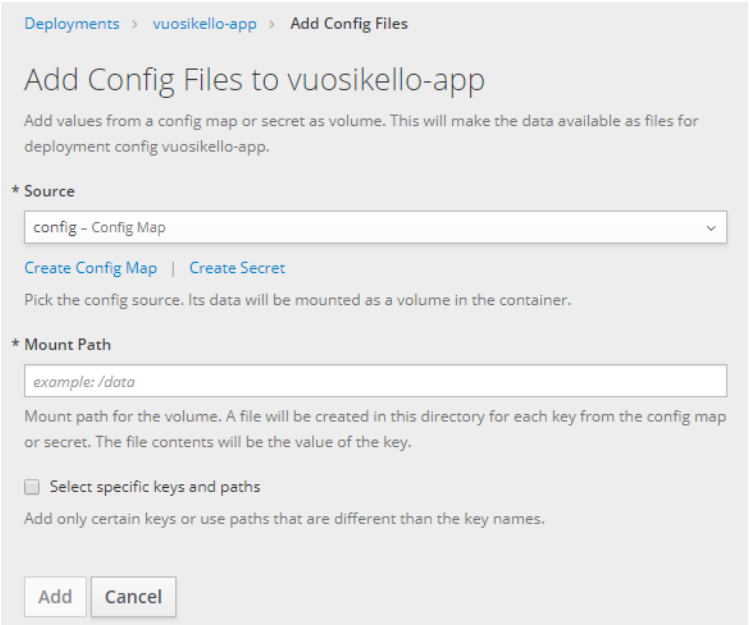
Kuva 14. ConfigMapin arvon injektoiminen ympäristömuuttajaksi (Kuvankaappaus OpenShift Online -käyttöliittymästä 20.2.2019)

Kuvassa 14 nähdään myös suoraan arvona injektoitava arvo ”KELLO”, jota käytetään tiedostopolkuna A/B-testauksessa luvussa 10. ConfigMap injektoidaan tiedostosta.

Tiedostoksi injektoiminen tapahtuu valitsemalla Applications-valikon kohdan ”Deployments” alta, välilehdeltä ”Configuration” ja sieltä Add Config Files (ympyröity kuvassa 15), joka avaa kuvan 16 dialoginäkymän, josta valitaan infektoitava ConfigMap (kuvat 15 ja 16).

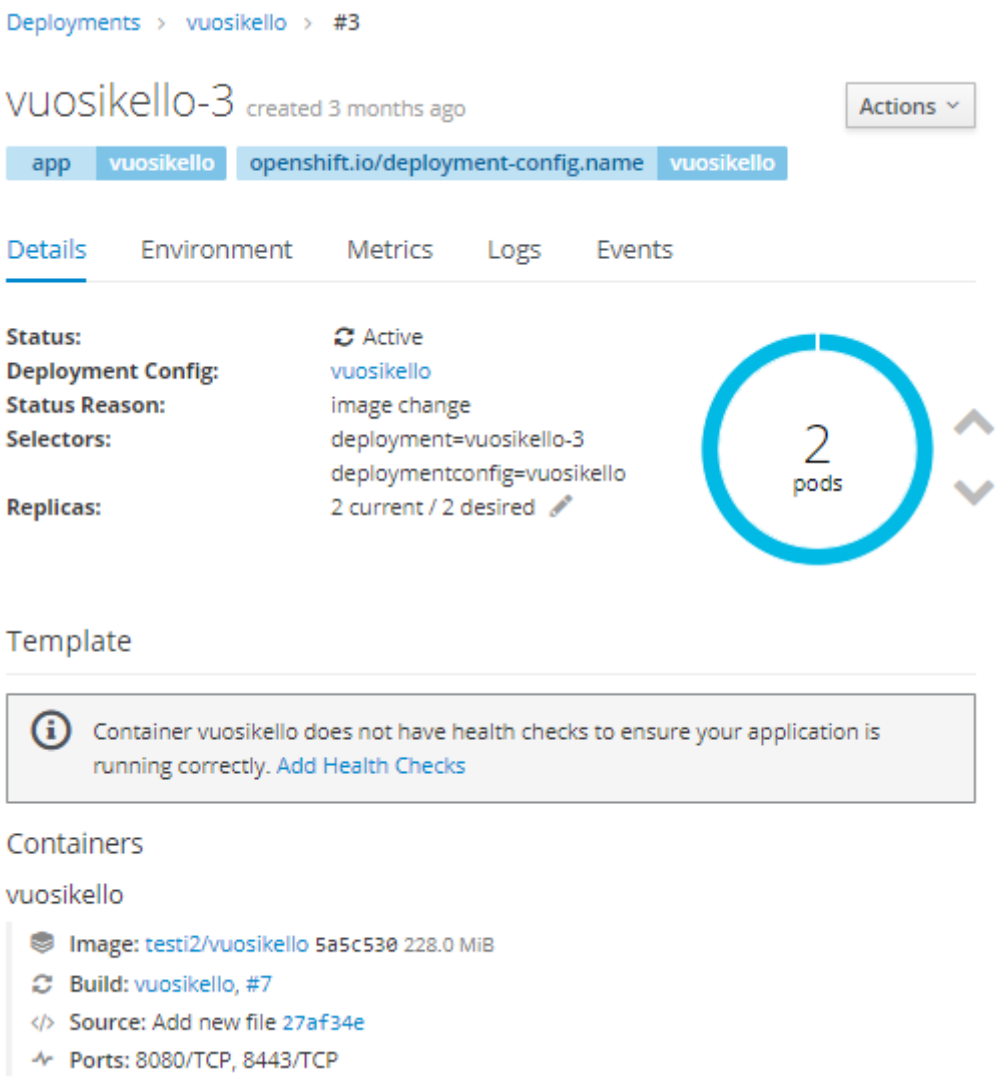


Kuva 15. ConfigMapin injektoiminen tiedostoksi. (Kuvankaappaus OpenShift Online -käyttöliittymästä 20.2.2019)



Kuva 16. ConfigMapin injektoimisen dialogi. (Kuvankaappaus OpenShift Online -käyttöliittymästä 20.2.2019)

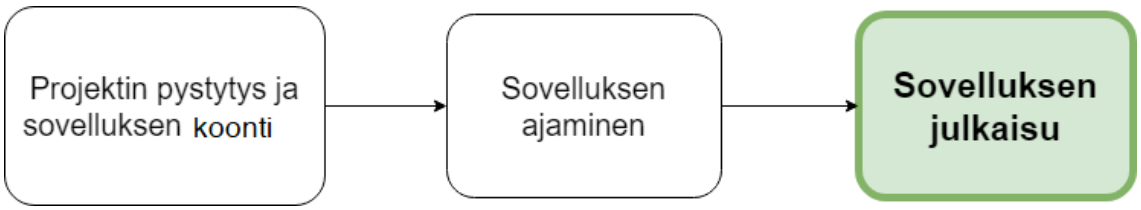
Kuvassa 16 on esitetty valmis, ja ajossa oleva kahden podin deployment OpenShift Online-käyttöliittymässä. Podien lukumäärää voidaan muuttaa lennosta ilman, että sovellusta täytyy ajaa alas klikkaamalla kuvan oikealla laidalla olevista nuolista. Luonnollisesti jos Podien määrän laskee nollaan, niin sovellus ajetaan alas. Kuvan infolaatikossa mainitut ”Health checkit” on selitetty luvussa 9.



Kuva 17. Valmiin deploymentin näkymä. (Kuvankaappaus OpenShift online-käyttöliittymästä 19.2.2019)

4.3 Sovelluksen julkaisu

Sovellus on tässä vaiheessa saatu koostettua ja ajettua, ja tämä luku keskittyy sen julkaisuun internettiin (kuva 18).

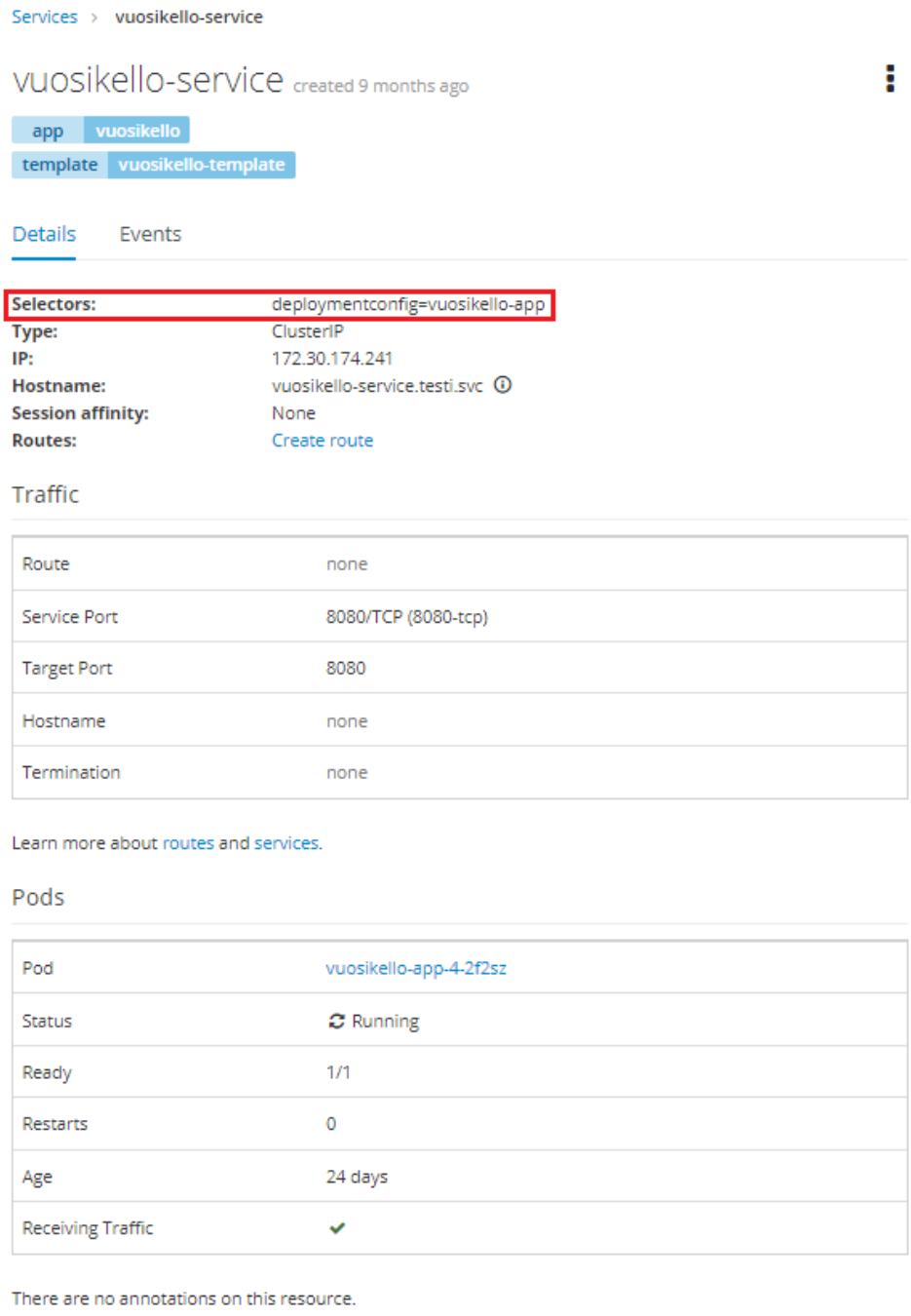


Kuva 18. Projektin työnkulku OpenShiftissä. Korostettuna meneillään oleva vaihe.

4.3.1 Servicet

Servicet ovat Kubernetesin mekanismi, jolla abstrahoidaan joukko Podeja yhden isäntänimen (engl. hostname) taakse OpenShiftin sisällä. (Termi Service jätetty suomentamatta, koska Kubernetesin käyttämä Service on oma käsitteensä, eikä sitä haluta sekoitettavan mihin tahansa yleiseen palveluun).

Service tasaa palvelulle tulevan kuorman Podeilleen. Hostnamen lisäksi Servicessä oleellista on valitsin (Selector). Valitsin määrittää, mille Podeille Serviceen tuleva liikenne ohjataan. Valitsimeksi asetetaan yksi tai useampi avain-arvo-pari. Serviceen valikoituvat ne Podit, joilla on etikettinä (Label) joku näistä pareista.

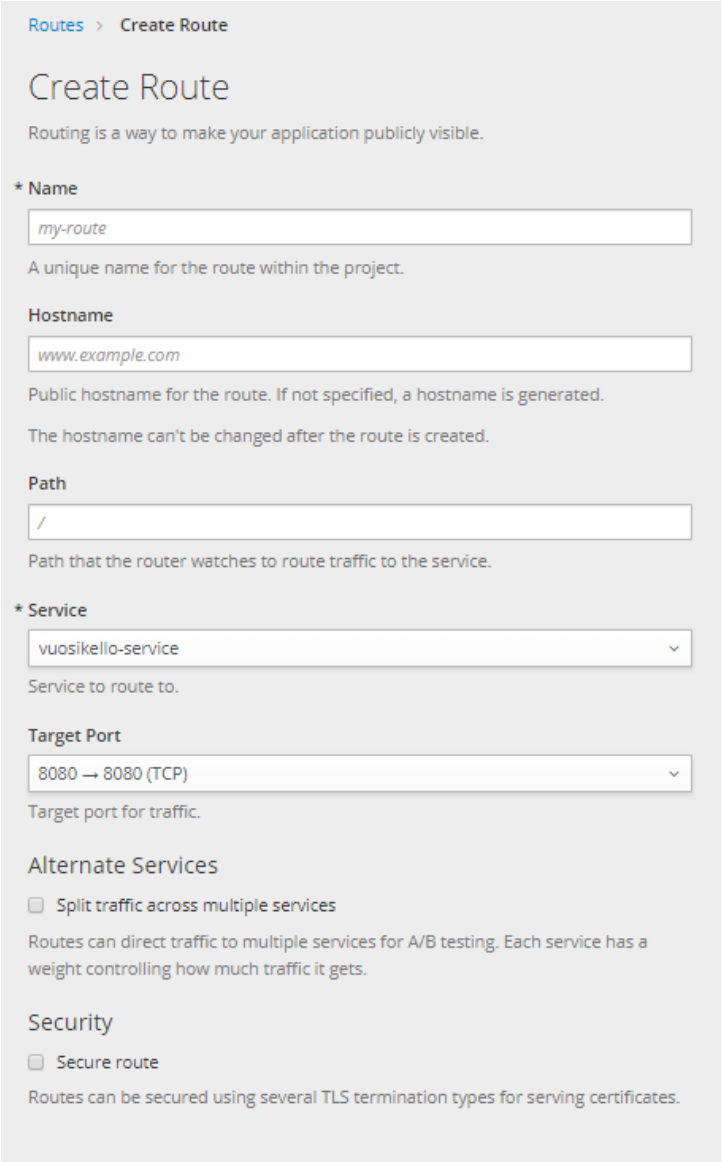


Kuva 19. Servicen asetukset. Kuvassa punaisella korostettuna Selector. Tähän Serviceen tulevat ne Podit, joilla Labelin deploymentconfig arvo on vuosikello-app. (Kuvankaappaus OpenShift Online -käyttöliittymästä 17.3.2019).

Yllä Kuvassa 19 nähdään OpenShift Onlinen -käyttöliittymässä Servicen julkaisemisen näkymä.

4.3.2 Reititys

Ulkoa tulevalle liikenteelle voidaan tehdä reititys (engl. route). Reitillä julkaistaan Service OpenShiftistä ulospäin näkyväksi sille määriteltyyn URL-osoitteeseen. Tässä yhteydessä voidaan päättää, julkaistaanko Service HTTP- vai HTTPS-protokollaa käyttäen (kuvassa 20 Security otsikon alla oleva valinta Secure route käyttää HTTPS-protokollaa).



Kuva 20. Reitityksen luomisen näkymä. (Kuvankaappaus OpenShift Online -käyttöliittymästä 17.3.2019).

5 Sovelluksen ylläpito ja A/B-testaus

Viimeisessä luvussa käydään läpi, kuinka OpenShiftillä voidaan ylläpitää sovellusta ja kuinka A/B-testausta tehdään.

5.1 Health checkit

DevOpsissa yleisesti ohjelmalle tavoitellaan jatkuvaa saatavuutta ja päivitykset halutaan tehdä siten, etteivät ne aiheuta pitkiä katkoksia palveluun.

OpenShiftin tarjoaa kaksi Kuberneteksen sovelluksen valvontaan tarkoitettua mekanismia, joita kutsutaan health checkeiksi. Nämä ovat nimiltään ”Readiness Probe” ja ”Liveness Probe”.

Readiness Probe testaa, onko kontti käynnistynyt deploymentin myötä oikein. Kun readiness Probe vastaa, niin OpenShift alkaa ohjaamaan sille liikennettä. Lisäksi mikäli Rolling Deployment on käytössä, niin Readiness Probe kertoo OpenShiftille, että se voi uuden kontin Deploymentin yhteydessä ajaa edellisen Deploymentin kontin alas.

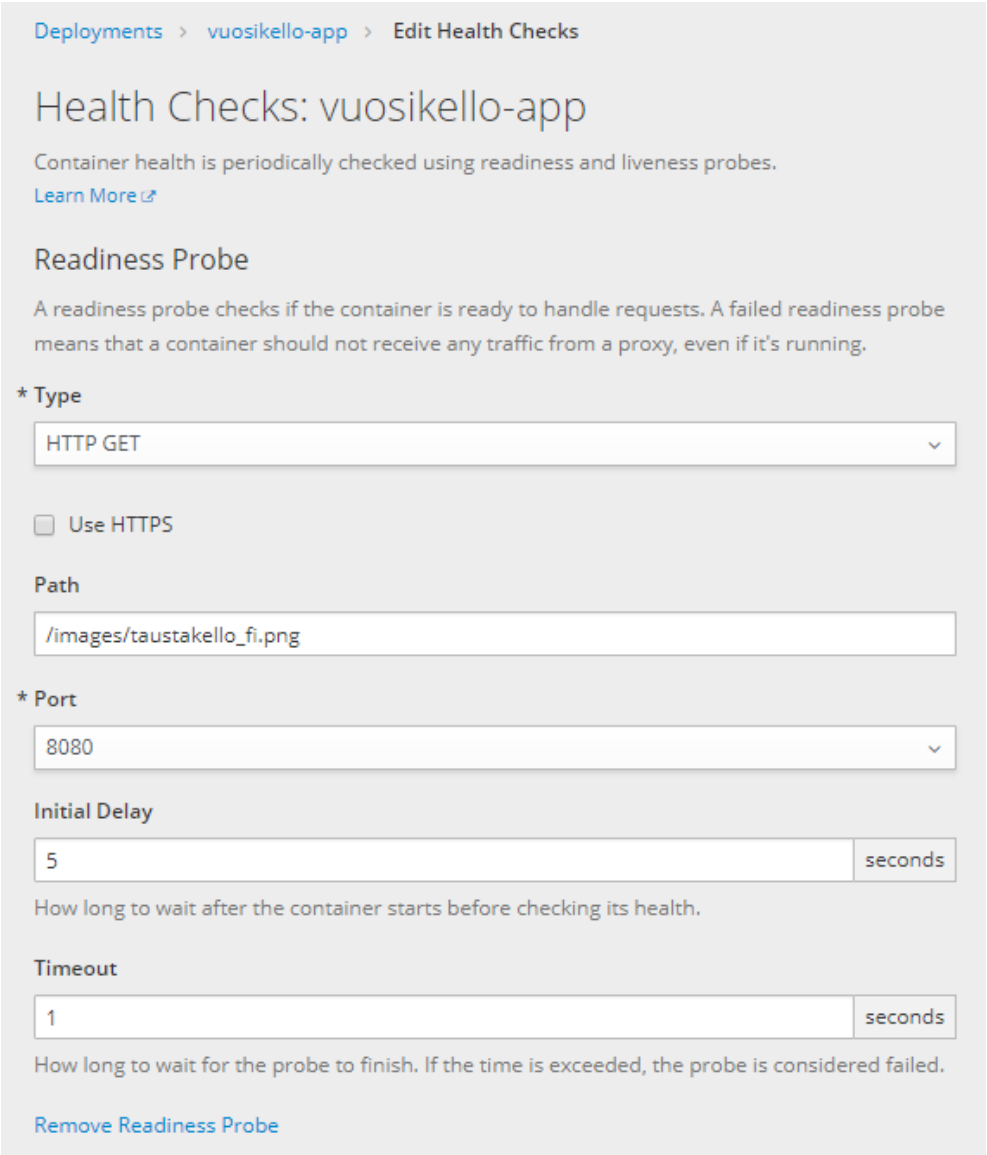
Liveness Probe puolestaan testaa vastaako kontissa ajettava sovellus yhä kutsuihin. Mikäli Liveness Probe epäonnistuu, OpenShift ei ohjaa kutsuja enää konttiin, vaan käynnistää sen uudelleen.

Health Checkit lisätään Application valikon kohdan Deployments alta. Mikäli Health Checkejä ei ole asetettu, niin OpenShiftin pitäisi näyttää kuvassa 17 Template otsikon alla näkyvä infolaatikko, jonka linkistä sen voi tehdä. Toinen tapa on valitsemalla Actions saman näkymän oikeasta yläkulmasta pudotusvalikosta ”Edit Health Checks”.

Avautuvasta näkymästä voidaan lisätä molemmat health checkit. Tähän projektiin käytettiin yksinkertaista HTTP GET -kutsua (muita testausvaihtoehtoja ovat Container Command ja TCP-Socket, mutta niihin ei tutustuttu tämän projektin puitteissa), joka lataa kontista kuvatiedoston. Probelle konfiguroidaan odotusaika, jonka jälkeen se aloittaa toimintansa. Sopiva aika täytyy selvittää kokeellisesti eli on testattava, kuinka kauan kontin pystyyn nousemisessa kestää sille annetuilla resursseilla.

Kuvassa 21 näkyvät Readiness Proben asetukset, jotka ovat identtiset Liveness Proben kanssa.

Probeilla on olemassa myös tärkeitä asetuksia, joita ei voi säätää tässä käyttöliittymässä, vaan ne täytyy laittaa sovelluksen YAML-konfiguraatioon. Näitä asetuksia ovat periodSeconds, jolla säädetään Proben ajamistiheyttä, ja successThreshold joka määrää, kuinka monta kertaa Probe saa epäonnistua.

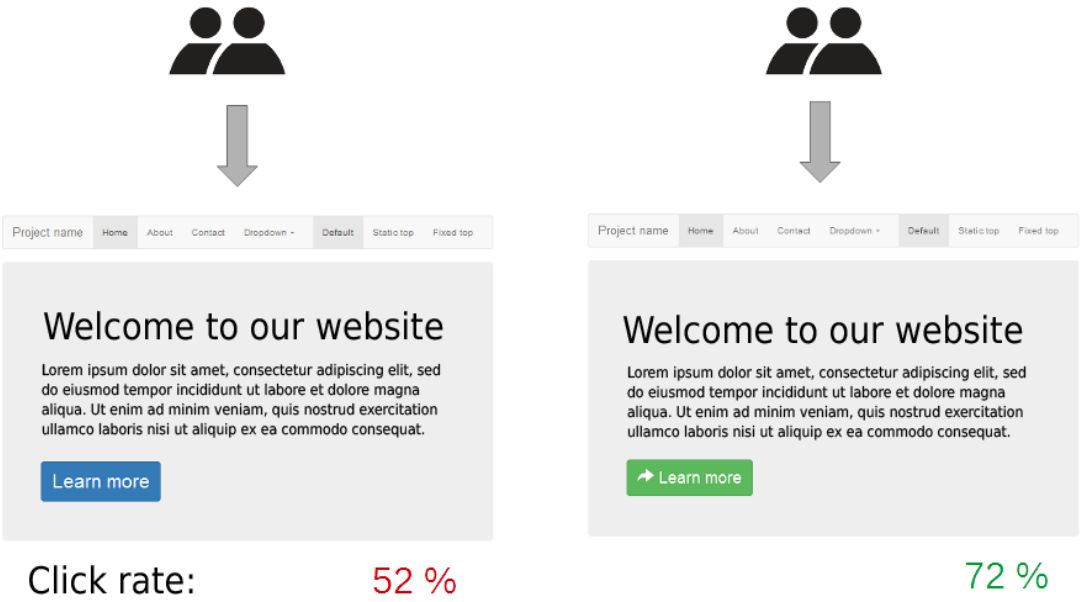


Kuva 21. Readiness Proben asetukset (Kuvankaappaus OpenShift Online käyttöliittymästä 20.2.2019).

Ajossa oleva Probe tekee GET-kutsuja, jotka näkyvät sovelluksen lokiselaimessa.

5.2 A/B-testaus

Työn viimeiseksi osuudeksi päätettiin testata vielä OpenShiftin A/B-testaukseen tarjoamat työkalut. A/B-testaamisella yleisesti tarkoitetaan tapaa tehdä paremmuusvertailua kahden eri vaihtoehdon välillä. Sovellusmaailmassa hyvä esimerkki on web-sivusto, jolle tultaessa on 50 %:n mahdollisuus, että sivun ulkoasu on tietynlainen ja 50 %:n mahdollisuus, että se on toisenlainen siitä huolimatta, että ne ovat samassa web-osoitteessa. (kuva 22).

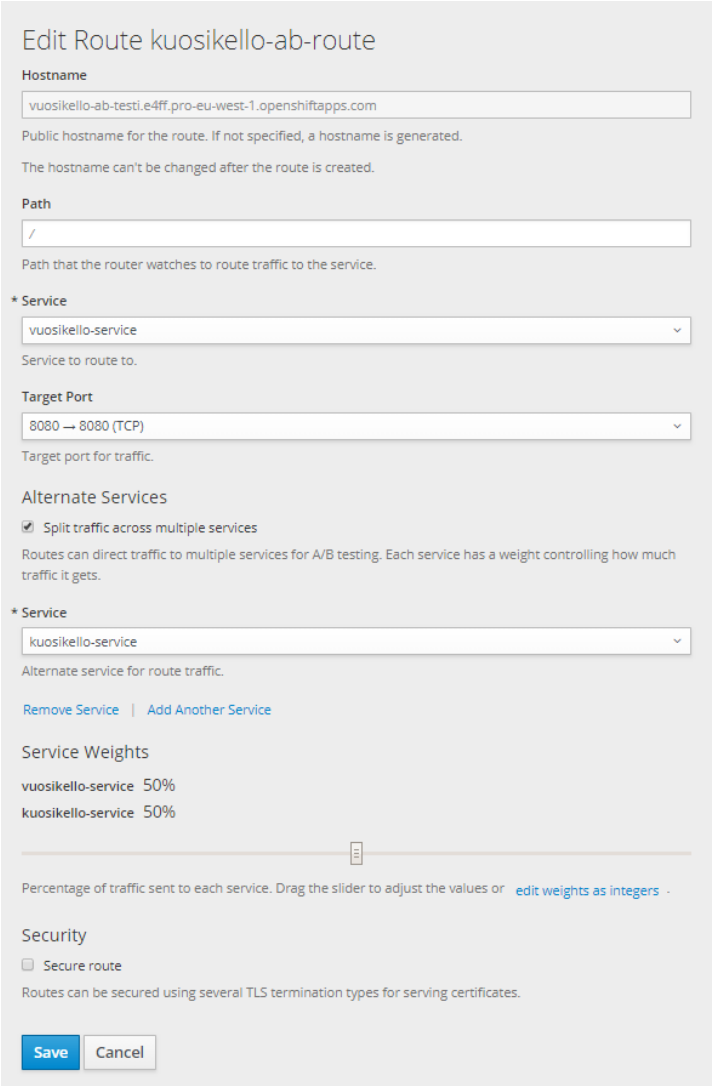


Kuva 22. Webbisivun A/B-testauksessa samasta sivustosta ajetaan eri versioita rinnakkain samassa osoitteessa. Kuvassa oikeanpuoleinen versio sivustosta vetoaa kävijöihin enemmän klikkausten perusteella. (https://en.wikipedia.org/wiki/A/B_testing#/media/File:A-B_testing_example.png CC-BY-SA 4.0 Luettu 19.2.2019)

Näin voidaan esimerkiksi vertailla sivuston eri versioiden käyttäjästatistiikkaa ja päättää, mikä versio otetaan pysyvästi käyttöön. OpenShift ei kuitenkaan tarjoa käyttäjästatistiikan työkaluja, mutta Deployment-kohtaisesti olisi mahdollista konfiguroida esim. Google Analytics.

OpenShiftissä A/B-testaaminen määritetään Route-tasolla. Routelle määritellään useampi Service sekä näiden suhteellinen painotus.

Alla kuvattuna (Kuvassa 21.) on näkymä, jolla A/B-reititys luodaan OpenShift Online käyttöliittymästä.



Kuva 23. Route laitetaan jakamaan kuormaa useammalle Servicelle valitsemalla ”Split traffic across multiple services”. (Kuvankaappaus OpenShift Online -käyttöliittymästä 17.3.2019).

Vuosikello-sovelluksesta luotiin OpenShiftiin toinen versio nimellä ”kuosikello” ja ainoa ero uuden ja vanhan version välillä on ympäristömuuttujan kautta injektoitava arvo vuosikellon kellotaulun grafiikan sisältävän kuvatiedoston polulle, joka ”kuosikello”-versiossa

sai räikeämmän värityksen. Näin saatiin aikaiseksi pienellä vaivalla kaksi eri näköistä versiota sivustosta A/B-testiä varten. Lopuksi siis kokeiltiin, miten A/B-testin saa toteutettua, mutta varsinaista testausta tämän projektin puitteissa ei tehdä.

6 Yhteenveto

Työssä perehdyttiin DevOpsin ajatusmalliin sekä useisiin teknologioihin, kuten Kubernetes ja Docker, joita käytetään DevOps-tyylisessä ohjelmistokehityksessä. Työn varsinainen tavoite oli testata OpenShift-ohjelmistoalustaa, joka myös hyödyntää em. teknologioita.

Työn alussa kartoitettiin OpenShiftin eri vaihtoehdot ja kokeiltiin OpenShiftin avoimen lähdekoodin version omatoimista asennusta, joka osoittautui liian haastavaksi ja aikaa vieväksi tehdä tämän työn puitteissa. Tämän vuoksi työhön valittiin pilvipalveluna tarjottava versio OpenShift-alustasta eli OpenShift Online.

OpenShift osoittautui erittäin monipuoliseksi ja päteväksi alustaksi, joka on suunniteltu juuri DevOps-tyyliseen ohjelmistojen kehitykseen, jossa pyritään minimoimaan manuaalinen työ sekä automatisoimaan mahdollisimman iso osa tehtävistä toimista. Siitä löytyvät ominaisuudet tekevät monesta melko mutkikkaasta asiasta, kuten palvelinresurssien allokoimisesta sitä tarvitsevalle ohjelmalle käyttäjän kannalta vaivatonta. Lisäksi palveluiden skaalaaminen ylös ja alas Podien määrää nostamalla ja laskemalla onnistuu nappia painamalla.

OpenShift hyödyntää vahvasti muita teknologioita kuten Kubernetesta, joka toimii sen konttien orkestroinnin työkaluna. OpenShift tarjoaa Kuberneteksen työkalujen lisäksi toiminnallisuudet palveluiden julkaisuun.

Vaikka projektin tuloksena ei syntynyt täydellistä CI/CD-putkea, niin OpenShift on omiaan sellaisen tekemiseen. Se olisi myös luonteva seuraava askel OpenShiftin testaamisessa.

Lähteet

1. The DevOps Handbook. Gene Kim, Jez Humble, Patrick Debois, John Willis. IT Revolution Press LLC. 2016.

2. IT Orchestration vs Automation: What's the Difference? https://www.bmc.com/blogs/it-orchestration-vs-automation-whats-the-difference/ Verkkoaineisto. Luettu 15.4.2019.

3. DevOps With OpenShift https://assets.openshift.com/hubfs/pdfs/DevOps_with_OpenShift.pdf?hsLang=en-us&extIdCarryOver=true&sc_cid=701f2000001OH7iAAG Stefano Picozzi, Mike Hepburn, Neil O'Connor. Luettu 15.3.2019.

4. OpenShift Documentation https://docs.openshift.com/container-platform/3.3/dev_guide/builds.html#configuring-the-source-environment. Verkkoaineisto. Luettu 15.3.2019.

5. OpenShift Documentation https://docs.openshift.com/enterprise/3.2/dev_guide/configmaps.html Verkkoaineisto. Luettu 22.2.2019.

Vuosi kellon koontikonfiguraation sisältävä YAML tiedosto


```yaml
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  creationTimestamp: '2018-06-19T19:19:12Z'
  labels:
    app: vuosikello-build
    template: vuosikello-build-template
  name: vuosikello-build
  namespace: testi
  resourceVersion: '258559622'
  selfLink: /apis/build.openshift.io/v1/namespaces/testi/buildconfigs/vuosikello-
  uid: a5fd484e-73f5-11e8-9f98-0645a759894e
spec:
  completionDeadlineSeconds: 1800
  nodeSelector: null
  output:
    to:
      kind: ImageStreamTag
      name: 'vuosikello-is:latest'
  postCommit: {}
  resources: {}
  runPolicy: Serial
  source:
    git:
      ref: openshift
      uri: 'https://gitlab.metropolia.fi/developers/vuosikello/vuosikello.git'
    sourceSecret:
      name: git-secret
    type: Git
  strategy:
    sourceStrategy:
      from:
        kind: ImageStreamTag
        name: 'php:7.1'
        namespace: openshift
    type: Source
  triggers:
    - type: ConfigChange
    - generic:
        secret: XXXXXX
      type: Generic
status:
  lastVersion: 4
```